Assignment 1 Report

Chan Jia Xin

ID: 31859089

Summary

My code uses the Model-View-Controller (MVC) architecture. When the user enters an input, my Controller which is the reduceState function will handle the input and return a new state. For example, if the user enters a 'w' key, the reduceState function will return a new state with the updated position of the frog. The Model is the current state of the game which stores all the data of the game, such as the score of the user, the position of the frog, the velocity of the planks, etc. The View component (updateView function) takes the data of the current state (Model) to present it to the users through creating or updating the SVG objects on the SVG canvas.

Design Decisions

Targets:

For each target, at least half of the frog must land on the circle to gain 100 points. To achieve this, I checked if the center of the frog is colliding with the target. If they are colliding with each other, the target is removed from the targets array and added to an array called collidedTargets. This target added has the same data except that the colour is changed to be same as the river, appearing as if it disappeared. Since it's no longer in the targets array, it will not be detected when handling collision, thus the frog dies when colliding with a target that has already been collided before.

Snakes:

The frog is allowed to land on the body of the snake, but not the head. It dies if it lands on the head and moves along with the snake when it's on the body. This is done by separating the snakes into two arrays, which are snakeHeads and snakeTails. The heads of the snakes have the same collision property as the cars, while the tails of the snakes have the same collision property as the planks.

Flies:

By eating (walking over) a fly, the player gains 50 points. The fly is initiated with random x and y positions, which changes randomly when the frog collides with it. These x and y positions are within the road section only.

How the Code Follows Functional Reactive Programming Style

By using observables, we can record asynchronous operations like user interface events in streams. In Functional Reactive Programming (FRP), we eliminate deeply nested loops and use just pure, referentially transparent functions. In my code, I used observables to capture user inputs for the movement of the frog, and to tick all the other objects in my game. Next, I also followed the FRP style by using functions such as map, filter, scan, etc. instead of loops throughout my code. For example, I used the map function to return a new array of planks with the new updated x positions. When handling collisions of cars, one of the functions I used was filter, to filter out the car that collided with the frog. The game ends when there is at least one car in the return array.

Purity

Maintaining purity is one of the key elements to Functional Reactive Programming. To adhere to Functional Reactive Programming, I maintained purity in my code whenever possible by keeping my functions pure. For instance, in my Tick or reduceState function, instead of mutating the current state to a new state, I created a new state with the updated data. I have utilized the built-in functions such as map and filter which creates a new array instead of mutating the input array.

Usage of Observables

- The **key$** stream is used to capture user's keyboard inputs.
- The **moveUp$, moveLeft$, moveDown$ and moveRight$** streams filter out the respective keys from key$ and map them to a new Move object with the respective x and y values.
- The **time$** stream produces a value every 10 milliseconds which is then mapped to a new Tick object.
- The moveUp$, moveLeft$, moveDown$, moveRight$ and time$ streams are merged into the **subscription$** stream which uses the scan function to update the state of the game. To display the current game state, I subscribed to subscription$ by passing in the updateView function.